Univerza v Ljubljani

Fakulteta za elektrotehniko

Uroš Šolar

Preizkus zmogljivosti in učinkovitosti kriptografske knjižnice Cryptography

Diplomsko delo visokošolskega strokovnega študija

Mentor: dr. Anton Umek, univ. dipl. inž. el.

Ljubljana, 2016

Zahvala

Zahvaljujem se viš. pred. dr. Antonu Umeku za izkazano potrpežljivost in podporo pri izdelovanju diplomske naloge. Prav tako, se zahvaljujem vsem članom laboratorija za Odprte sisteme in mreže na Inštitutu Jožefa Štefana, še posebno pa dr. Dušanu Gabrijelčiču za vodstvo in priložnost za sodelovanje na projektu "Flex4Grid," v sklopu katerega je nastala tudi diplomska naloga.

Vsebina

Seznam tabel

Seznam slik

Seznam odsekov programske kode

Seznam uporabljenih simbolov

V zaključnem delu so uporabljene naslednje veličine in simboli:

Veličina / oznaka		Enota	
Ime	Simbol	Ime	Simbol
čas	t	sekunda	s
dolžina bloka	L_{blok}	oktet/bajt	B
pretok	P_{blok}	bajt na sekundo	$\left[\frac{B}{s}\right]$
operacije	O	št. operacij na sekundo	$\left[\frac{op}{s}\right]$
pretok posamezne funkcije (cryptography) za določen vhodni blok	${P^{pyc}}_{blok}$	bajt na sekundo	$\left[\frac{B}{s}\right]$
pretok posamezne funkcije (openssl) za določen vhodni blok	${P^{ossl}}_{blok}$	bajt na sekundo	$\left[\frac{B}{s}\right]$
učinkovitost	η	/	/

Tabela 0.1: Veličine in simboli

Povzetek

V nalogi se predstavi programska knjižnica Cryptography za programski jezik Python, ki deluje kot programska ovojnica za kriptografske funkcije implementirane v nižje-nivojskih programskih jezikih. Za uvod k prikazu preizkusa zmogljivosti in učinkovitosti funkcij služi predstavitev izbranih kriptografskih mehanizmov knjižnice kot so SHA-2, AES, RSA in ECDSA, ki se predstavijo v kontekstu osnovnih varnostnih lastnosti. Prikaz funkcij knjižnice Cryptography predvsem izpostavi estetiko in uporabo programske ovojnice kriptografskih funkcij.

Zmogljivost obeh programskih knjižnic se meri z merilom bitne pretočnosti in številom operacij na sekundo pri omenjenih funkcijah, učinkovitost pa se meri kot kvocient zmogljivosti knjižnice Cryptography in OpenSSL. Primerjava obeh metrik nam omogoča, da se v nalogi razišče prednosti in slabosti uporabe knjižnice Cryptography ter ustvari sklep o potencialni uporabi knjižnice v spletnih aplikacijah.

Izračun učinkovitosti nam pove, da se hitrost računanja funkcij knjižnice Cryptography, najbolj približa hitrosti funkcij iz knjižnice OpenSSL pri računsko intenzivnejšimi postopki. Pridemo do zaključka, da glavna prednost programske knjižnice Cryptography izhaja iz neodvisnosti uporabe številnih zalednih knjižnic in preproste programske ovojnice.

Ključne besede: programska knjižnica Cryptography, Python, preizkus kriptografskih funkcij

Abstract

The thesis presents the Cryptography programming library for the Python programming language which serves as a wrapper for the cryptographic functions implemented in low-level programming languages. The presented cryptographic procedures such as SHA-2, AES, RSA and ECDSA serve as a basis for the demonstration of their performance and efficiency in comparison with the functions of the OpenSSL programming library. Moreover, the presentation also discusses the security properties each family of functions provide. Additionally, the demonstration highlights the esthetics and usage of the cryptographic wrappers.

The performance of functions from both programming libraries is measured by calculating the throughput and the number of operations per second. In addition, efficiency is also calculated by taking a quotient of performance results from functions of the Cryptography and OpenSSL programming libraries. The comparison of both metrics gives the ability to form an informed conclusion about the potential uses for the library. It shows that the performance of Cryptography's functions is the closest to the functions from the OpenSSL library when computationally intensive functions are used. We come to the conclusion that the main advantage of the Cryptography programming library comes from the independent use of the different back-end libraries and its simple function wrappers.

Key words: programming library Cryptography, Python, performance of cryptographic functions

1 Uvod

V času, kjer je informacijska varnost postala pomembna tematika slehernega posameznika družbe, se od razvijalcev računalniških storitev pričakuje, da zagotavljajo določene varnostne vidike. Od storitev spletnega bančništva ali spletnega trgovanja, kjer je nevarnost zlorabe visoka, do preprostih spletnih strani.

Zaradi težavnosti načrtovanja in implementacije kriptografskih postopkov je priporočljivo, da razvijalci storitev posežejo po programskih knjižnicah kot je OpenSSL[1], ki vsebuje že preverjenje kriptografske algoritme in za katere se domneva, da zagotavljajo dovolj visoko mero varnosti [1]. Na žalost pa so ponavadi mehanizmi delovanja takih knjižnic v večini slabo dokumentirani, saj predpostavljajo določeno mero znanja, in predvsem zrelosti pri uporabi. Prav tako se pri večini kriptografskih knjižnicah predpostavi znanje določenih t.i. "nižje ležečih" programskih jezikov kot so *C* ali *C++*.

V nadeljevanju se predstavi kriptografska knjižnica *Cryptography* za programski jezik Python [2]. Knjižnica Cryptography služi kot ovojnica za uveljavljene knjižnice kot so OpenSSL ali Common Crypto.[2] Obe knjižnici sta ključni del protokolov kot so SSL[3] (*ang. Secure Sockets Layer*) in TLS[4] (*ang. Transport Layer Security*). Razvijalcem ponuja prijazen način dostopa do nižje ležečih kriptografskih funkcij v jeziku Python. Python je predvsem znan po preprostosti uporabe in berljivosti programske kode, zato se preprosta sintaksa ovojnice še posebej izpostavi. Opiše se tudi osnovne varnostne lastnosti, ki služijo za pojasnitev vlog različnih kriptografskih funkcij.

V jedru naloge stoji primerjava zmogljivosti kriptografskih funkcij iz knjižnic OpenSSL in Cryptography. Zmogljivost se meri s številom operacijami na sekundo ali bitnim pretokom posameznih funkcij. Iz primerjave zmogljivosti, pa se prav tako

[1] Odprto-kodna kriptografska knjižnica implementirana v programskem jeziku C.
[2] Kriptografska knjižnica, ki jo podpira Apple in se uporablja v njihovih produktih.
[3] Starejši kriptografski protokol, ki se je predhodnik protokola TLS.
[4] Kriptografski protokol, ki vsebuje nabor šifer in postopkov potrebnih za varno IP komunikacijo.

izpostavi tudi učinkovitost programske knjižnice Cryptography v primerjavi z neposredno implementacijo kriptografskih funkcij v knjižnici OpenSSL. S pomočjo merila učinkovitosti bo v zaključnih poglavjih podano mnenje o potencialu uporabe, ki jo ima knjižnica Cryptography.

1.1 Obseg naloge

Preizkus je opravljen na računalniku s procesorjem Intel i5 4200u in 4GB delovnega spomina. Prav tako, se omeji nabor programske opreme, saj se preizkusi OpenSSL 1.0.1f in Cryptography 1.3.1 pri različici Python 3.4.3. Ob predstavitvi kriptografskih funkcij knjižnice izberemo samo najbolj osnovne predstavnike kriptografskih postopkov kot so zgoščevalne funkcije in simetrične ter asimetrične funkcije.

Količine, ki se merijo pri preizkusu zmogljivosti so izbrane na podlagi sorodnih kriptografskih preizkusih [3]. Večinoma se zgleduje po hitrostnem preizkusu programske knjižnice OpenSSL (*openssl speed*). Oblika rezultatov nam omogoča lažjo primerjavo s podobnimi preizkusi ostalih kriptografskih knjižnic.

2 Osnovni varnostni model

Za razumevanje različnih vlog, ki jih izpolnjujejo kriptografske funkcije predstavljene v naslednjih poglavjih, je potrebno definirati osnovne varnostne lastnosti in pojme. Osnovni varnostni model določa varnostne cilje, ki jih kriptografske funkcije zagotavljajo.

V stroki pogosto prihaja do nestrinjanja pri definiranju osnovnega pojma informacijske varnosti, saj nekateri raziskovalci pri tem zajemajo celoten informacijski sistem, ostali pa samo podatke. Starejša definicija informacijske varnosti kot CIA[5] (*ang. Confidentiality, Integrity, Availability*) triade ni več dovolj, saj se v prevzema pogled na informacijsko varnost kot proces varovanja intelektualne lastnine organizacije [4]. Kljub temu, se v tem delu predstavi osnovni model informacijske varnosti iz vidika varovanja podatkov, saj programska knjižnica Cryptography ni orodje za določanje varnostne politike organizacij.

Analiza različnih varnostnih scenarijev nam omogoča prepoznavo naslednjih ciljev, ki jih izpolnjuje vsak varen komunikacijski kanal.

- **Zaupnost** preprečuje branje podatkov vsem, ki nimajo avtorizacije za branje.
- **Verodostojnost** poskrbi, da se podatki ne spreminjajo brez primerne avtorizacije pošiljatelja.
- **Avtentičnost** zagotavlja izjavljeno identiteto pošiljatelja
- **Neovrgljivost** preprečuje komunikacijskem osebku zanikanje preteklih sporočil ali dejanj.

Nekatere aplikacije imajo tudi dodatne varnostne cilje kot so anonimnost komunikacijskih osebkov ali nadzor dostopa (*ang. access control*) osebkom. Na osnovne varnostne lastnosti se sklicuje pri predstavljanju osnovnih gradnikov knjižnice Cryptography.

[5] Osnovni varnostni model, ki ga sestavljajo lastnosti: Zaupnost, Verodostojnost, Avtentičnost.

3 Programska knjižnica Cryptography

Programska knjižnica Cryptography je odprtokodna zbirka kriptografskih primitivov in postopkov, ki jih lahko dostopamo v programskem jeziku Python. Vsebuje uveljavljene kriptografske algoritme, s katerimi se gradi kriptografske mehanizme in protokole za izdelavo varnih računalniških sistemov. Za knjižnico je značilen API,[6] (ang. *application programming interface*) ki je lahko berljiv in prijazen za uporabo.

Knjižnica ne implementira lastnih kriptografskih primitivov ampak v zaledju uporablja uveljavljene implementacije algoritmov v programskih jezikih C ali C++. Ena izmed glavnih prednosti knjižnice je uporaba ene ali več programskih knjižnic hkrati za dostop do nižje ležečih kriptografskih algoritmov, ki so implementirani v uveljavljenih knjižnicah sta OpenSSL ali CommonCrypto.

CommonCrypto se uporablja v aplikacijah, ki delujejo na programski opremi podjetja Apple. Platforme kot so iOS[7] in Mac OS X[8]. V še večji meri se uporablja knjižnico OpenSSL, saj je privzeta kriptografska knjižnica strežnikov Apache[9] in nginx[10], ki pa skupaj sestavljata do 66% vseh strežnikov na spletu [5].

Cryptography sestavljajo dve različne abstraktne plasti, ki jih lahko dostopajo uporabniki. Najnižja plast pa je namenjena razvijalcem knjižnice, saj vsebuje vezi do različnih kriptografskih funkcijah, ki so implementirane v programskem jeziku C ali C++. Prva abstraktna plast se imenuje hazmat[11] (*ang. hazardous materials*) in omogoča dostop do kriptografskih primitivov prek programske ovojnice v Pythonu. Pri izdelavi preizkusa se uporabi funkcije plasti hazmat zato se tudi pri predstavitvi nameni največ pozornosti tej plasti.

[6] Programska ovojnica
[7] Operacijski sistem namenjen prenosnim napravam družine iPhone in iPad.
[8] Operacijski sistem namenjen namiznim in prenosnim računalnikom podjetja Apple.
[9] Najbolj razširjena programska oprema za delovanje spletnega strežnika.
[10] Druga najbolj razširjena programska oprema za delovanje spletnega strežnika.
[11] Abstraktna plast kriptografskih primitivov

Druga plast se imenuje recepti (*ang. recipes*), ki predstavi programsko ovojnico z idejo, da za zagotavljanje varnostih lastnosti uporabnik potrebuje le vhodni čistopis. Vse ostale podrobnosti, kot izbira mehanizmov delovanja ali pridelava vrednosti IV, ki potrebujejo domensko znanje kriptografije, se skrijejo za ustrezno abstrakcijo. Primer take abstrakcije je šifra Fernet, ki uporablja mehanizem AEAD[12] (*ang. Authenticated Encryption with Associated Data*). Fernet deluje na osnovi bločne šifre, ki poleg zaupnosti podatkov zagotavlja tudi verodostojnost in avtentičnost.

Bistvo vsega je, da uporabnik plasti recipes ne potrebuje specifičnega znanja za uporabo preprostih gradnikov ampak za ustrezno zagotavljanje varnostih lastnosti poskrbi nivo abstrakcije

3.1 Osnovni gradniki

Osnovni gradniki ali kriptografski primitivi knjižnice Cryptography se nahajajo na plasti "hazmat." Plast se razdeli na različne funkcionalnosti, saj vsebuje zgoščevalne funkcije, simetrične in asimetrične kriptografske postopke, bitno zapolnjevanje, (*ang. padding*) itd.

Glavni gradnik knjižnice je CFFI[13] (*ang. Common Foreign Function Interface*), ki v primeru programske knjižnice Cryptography omogoča komunikacijo med OpenSSL in programsko ovojnico jezika Python.

V Odsek programske kode 1 prikažemo klic modulov, ki so potrebni za izvajanje kriptografskih funkcij. Za potrebe naloge, se predstavijo funkcije iz Cryptography dokumentacije, ki so uporabljene v preizkusu zmogljivosti.

```
from Cryptography.hazmat.backends import default_backend
from Cryptography.hazmat.primitives import hashes
from Cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from Cryptography.hazmat.primitives.asymmetric import padding, rsa, ec
```

Odsek programske kode 1: Osnovni moduli plasti hazmat

Poleg modulov, ki so potrebni za uporabo kriptografskih primitivov pokličemo tudi hrbtenični modul (*ang. default_backend*), ki skrbi za povezavo med Python API in knjižnico OpenSSL.

Predstavljene funkcije programske knjižnice so izvzete iz preizkusa zmogljivosti zato je programska ovojnica definirana v kontekstu preizkusa.

[12] Kriptografski postopek, ki ob enem zagotavlja tajnost, verodostojnost in avtentičnost podatkov.

[13] Mehanizem za dostopanje do funkcij in spremenljivk drugega programskega jezika.

3.2 Zgoščevalne funkcije

Zgoščevalne funkcije pridelajo zgoščeno vrednost (digitalni prstni odtis) fiksne dolžine iz poljubno dolgega niza bitov. Pomembno je, da isti niz vhodnih podatkov vedno pridela enako vrednost, in da nihče ne more zanesljivo sestaviti dveh različnih sporočil, ki bi imela enak digitalni prstni odtis. Zgoščevalne funkcije so ključni gradnik MAC[14] (*ang. message authentication code*) in MDC[15] (*ang. modification detection code*) funkcij, s katerimi se zagotavlja verodostojnost, avtentičnost in neovrgljivost podatkov [6].

V Odsek programske kode 2 je prikazana osnovna struktura klica zgoščevalne funkcije v programski knjižnici Cryptography.

```
def SHA_hash(self, sha_type, input_blck):
    digest = hashes.Hash(sha_type(), backend=default_backend()) #Primer: sha_type = hashes.SHA256
    digest.update(input_blck)
    digest.finalize()
```

Odsek programske kode 2: Zgoščevanje s SHA-2

Odsek je del testa zmogljivosti, kjer funkcija pričakuje za vhodni parameter zgoščevalni primitiv družine SHA-2[16] (*ang. Secure Hash Algorithm*) in vhodni podatkovni blok. Metoda *update* določi vhodni čistopis, ki ga je potrebno zgostiti. Na koncu metoda *finalize* poskrbi, da se čistopis neodvisno zgosti v skladu z določeno zgoščevalno funkcijo (v tem primeru SHA-256). Prav tako poskrbi za zaključitev trenutnega zgoščevalnega konteksta.

SHA-2 standard sestavljata dve zgoščevalni funkciji SHA-256 in SHA-512. Temeljita na konstrukciji Davies-Meyer, ki deluje kot bločni šifrirni postopek, kjer se vsak trenutni blok sporočila uporabi kot ključ za šifriranje prejšnje zgoščene vrednosti. Na koncu se trenutna šifrirana vrednost in prejšnja zgoščena vrednost uporabita v operaciji XOR[17] (*ang. exclusive or*).

Funkciji družine SHA-2 se razlikujeta pri računanju v številu ponovitev in velikosti bitne besede. SHA-256 uporablja pri računanju 32-bitne besede s 64 ponovitvami, SHA-512 pa 64-bitne besede z 80 ponovitvami. V komentarju je naveden primer testa za funkcijo SHA-256.

[14] Kode za zagotavljanje avtentičnosti sporočila.
[15] Kode za detekcijo spremembe sporočila.
[16] Kriptografska zgoščevalna funkcija organizacije NSA.
[17] Operacija ekskluzivne disjunkcije

3.3 Simetrična kriptografija

Simetrično šifriranje zagotavlja zaupnost informacije med komunikacijskimi osebki, ki poznajo vrednost skrivnega ključa. Postopek preslika vrednost čistopisa (*ang. plaintext*) v neberljiv kriptogram (*ang. ciphertext*). Govorimo o simetričnem šifriranju, ker komunikacijski osebki uporabljajo enak ključ za šifriranje in dešifriranje podatkov. Zaupnost šifriranje informacije temelji na skrivni vrednosti ključa, saj sta algoritma za enkripcijo in dekripcijo podatkov javna.

Šifrirne algoritme ločimo na bločne in pretočne šifre. Bločne šifre operirajo z vhodnim čistopisom, tako da se razdeli na določeno število blokov fiksne dolžine. Pretočne šifre pa operirajo z vhodnim čistopisom kot s posameznimi simboli.

V preizkusu se uporabi simetrična šifra AES (*ang. advanced encryption standard*) v CBC[18] (*ang. cipher block chaining*) načinu, ki deluje kot bločna šifra dolžine 128 bitov. AES je sestavljen z večimi ponovitvami postopkov, kjer se uporabljajo operacije substitucije in permutacije. Postopki so obrnljivi in število ponovitev je odvisno od dolžine skrivnega ključa; 10 ponovitev za ključ dolžine 128 bitov, 12 ponovitev za ključ dolžine 192 bitov in 14 ponovitev za ključ dolžine 256 bitov [7].

Implementacija AES algoritma v Cryptography je prikazana v Odsek programske kode 3.

```
def AES_encrypt(self, key, input_blck):
    iv = os.urandom(16)
    cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend = default_backend())
    encryptor = cipher.encryptor()
    encryptor.update(input_blck)
    encryptor.finalize()
```

Odsek programske kode 3: Enkripcija z AES

AES v CBC načinu potrebuje začetno vrednost IV[19] (*ang. Initialization Vector*), saj ta poskrbi, da se enaki čistopisi preslikajo v različne kriptograme. IV je ponavadi naključno pridelana vrednost, ki se pošlje ob kriptogramu kot čistopis. Struktura implementacije šifre AES je podobna zgoščevalni strukturi, saj neodvisno sprejme vhodne podatke in zaključi šifriranje z enako poimenovanimi *update*, *finalize*

[18] Način šifriranja, kjer se predhodni šifrirani blok uporabi kot vhod za šifriranje trenutnega bloka.

[19] Vhodna informacija pri šifriranju prvega bloka v CBC načinu.

metodami. Razlika se najde v strukturi začetnega *Cipher* objekta, ki sprejme poleg tipa algoritma in skrivnega ključa *key*, tudi način delovanja *modes.CBC(iv)*.

Skrivni ključ se pridela predhodno s pomočjo funkcije os.urandom, ki ustvari naključno zaporedje bitov specificirane dolžine.

3.4 Asimetrična kriptografija

Asimetrično kriptografijo definira koncept deljenih kriptografskih ključev. Postopki asimetrične kriptografije uporabljajo javni in zasebni ključ za zagotavljanje lastnosti avtentikacije in zasebnosti. Ena izmed glavnih nalog asimetrične kriptografije ali kriptografije z javnim ključem, pa je varna distribucija skrivnih ključev za simetrične kriptografske postopke.

V kriptografski shemi z javnim ključem ima vsak od komunikacijskih osebkov svoj javni in zasebni ključ. Postopek je zasnovan tako, da čistopis šifriramo z javnim ključem naslovnika in dešifriramo z zasebnim ključem. Bolj pogosteje se pa uporablja asimetrične postopke za digitalno podpisovanje, kjer se zgoščena vrednost sporočila najprej šifrira s privatnim ključem pošiljatelja in preveri z javnim ključem na strani naslovnika. Visoka računska cena asimetričnih postopkov je glavni razlog zakaj se asimetrične postopke ne uporablja pogosteje za neposredno šifriranje podatkov.

V preizkusu knjižnice Cryptography se uporabita dva asimetrična algoritma; RSA (*ang. Rivest, Shamir, Adleman*) in ECDSA (*ang. Elliptic Cruve Digital Signing Algorithm*).

RSA temelji na težavnosti faktorizacije velikih števil, ki omogoča sestavljanje enosmernih funkcij s stranskimi vrati (*ang. one-way trapdoor functions*) [8]. Verifikacija RSA sporočil je zaradi načrtne odločitve iznajditeljev RSA postopka veliko hitrejša od podpisovanja. To dejstvo se skrbno uporablja pri načtrovanju različnih kriptografskih protokolov, kjer se verifikacija opravlja na počasnejši strojni opremi ali v večjem številu.

Proces pridelave ključa je v knjižnici Cryptography prikazan v Odsek programske kode 4.

```
def RSA_key_gen(self, key_size):
    private_key = rsa.generate_private_key(public_exponent=65537, key_size=key_size,
                                           backend=default_backend()
    )
    return private_key, private_key.public_key()
```

Odsek programske kode 4: Pridelava javnih in zasebnih ključev RSA

Postopek pridelave ključev se opravi z metodo *rsa.generate_private_key*. Parameter *key_size* določimo po potrebi aplikacije. Ključi dolžine 1024 bitov so v teoriji še vedno varni, ampak se za nove aplikacije priporoča uporaba ključev dolžine 2048 in 4096 bitov. Parameter *public_exponent* določa velikost javnega eksponenta, ki se uporabi pri izračunu kriptograma. Standardizirana vrednost javnega eksponenta je 65537, saj more izbrana vrednost izpolnjevati številne pogoje [8].

Funkcija nam vrne pridelan par ključev, ki se uporabljajo pri podpisovanju in verifikaciji. Uporabljeni funkciji sta prikazani v Odsek programske kode 5 in Odsek programske kode 6.

```
def RSA_signing(self, private_key, input_blck):
    signer = private_key.signer(padding.PSS(
            mgf=padding.MGF1(hashes.SHA256()),
            salt_length=padding.PSS.MAX_LENGTH
        ),
        hashes.SHA256()
    )
    signer.update(input_blck)
    return signer.finalize()
```

Odsek programske kode 5: Podpisovanje pri RSA

RSA podpisovanje potrebuje določeno zgoščevalno funkcijo in funkcijo za bitno zapolnjevanje. PSS[20] (*ang. Probabilistic Signature Scheme*) pričakuje vrednost *mgf* (*ang. mask generator function*), ki iz okteta podatkov deterministično pridela niz bitov in *salt_length*, ki služi za pridelavo vhodnega niza za funkcijo *mgf* [9].

```
def RSA_verification(self, public_key, signature, input_blck):
    verifier = public_key.verifier(signature, padding.PSS(
                mgf=padding.MGF1(hashes.SHA256()),
                salt_length=padding.PSS.MAX_LENGTH
                ),
            hashes.SHA256()
            )
    verifier.update(input_blck)
    verifier.verify()
```

Odsek programske kode 6: Verifikacija podpisov pri RSA

[20] Tip bitnega zapolnjevanja, ki se uporablja pri podpisovanju in verifikaciji sporočil RSA.

Iz zgornjega programskega odseka je razvidna podobnost pričakovanih parametrov verifikacije s procesom podpisovanja. Knjižnica uporablja mehanizem izjeme (*ang. exception*) za obveščanje o detekciji napačnih podpisov.

Drug, že omenjen preizkušen asimetrični postopek je ECDSA. Algoritem enako služi podpisovanju in verifikaciji sporočil, vendar temelji na težavnosti drugega matematičnega problema. Iz težavnosti reševanja problema diskretnega logaritma eliptičnih krivulj lahko prav tako izdelamo enosmerno funkcijo s stranskimi vrati, ki je osnova za algoritem ECDSA [10].

Kriptografska skupnost je v preteklosti imela več uspeha z reševanjem problemov, ki temeljijo na celoštevilski faktorizaciji v primerjavi s problemi diskretnega logaritma eliptičnih krivulj, zato ECDSA zagotavlja veliko višjo stopnjo varnosti pri ključih enakih velikosti kot RSA. Kar pa pomeni, da je ECDSA pri enaki stopnji varnosti veliko hitrejši od postopka RSA. To je tudi razvidno iz preizkusa zmogljivosti.

Varnost in hitrost ECDSA je odvisna od izbrane eliptične krivulje. Nemalo število kriptografskih krivulj je patentirano zato se pri izbiri krivulj poslužujemo standardiziranih funkcij iz dokumentov NIST FIPS 186-2 [11]. Izbrani krivulji sta secp256r1 in secp384r1. Obstajajo določeni zadržki o možnem obstoju stranskih vrat (*ang. backdoor*) pri krivuljah tipa r1, ki so standardizirane s strani organizacij kot so NSA (*ang. National Security Agency*) in NIST (*ang. National Institute of Standards and Technology*) [12].

V Odsek programske kode 7 vidimo proces pridelave ECDSA javnih in zasebnih ključev v knjižnici Cryptography.

```
def ECDSA_key_gen(self, ec_type):
    private_key = ec.generate_private_key(ec_type, default_backend()) #primer ec_type: ec.SECP384R1
    return private_key, private_key.public_key()
```

Odsek programske kode 7: Pridelava javnih in zasebnih ključev ECDSA

Naslednji prikaz v Odsek programske kode 8 vsebuje proces podpisovanja in verifikacije.

```
def ECDSA_signing(self, private_key, input_blck):
    signer = private_key.signer(ec.ECDSA(hashes.SHA256()))
    signer.update(input_blck)
    return signer.finalize()


def ECDSA_verification(self, public_key, signature, input_blck):
    verifier = public_key.verifier(signature, ec.ECDSA(hashes.SHA256()))
    verifier.update(input_blck)
    verifier.verify()
```

Odsek programske kode 8: Podpisovalne in verifikacijske funkcije ECDSA

Edina opazljiva razlika med ECDSA in RSA podpisovalnim – verifikacijskim API, je v tem, da ECDSA ne potrebuje funkcije bitnega zapolnjevanja, saj je velikost zgoščene vrednosti, ki se podpisuje zmeraj manjša ali enaka velikosti vrednosti zasebnega ključa.

4 Izdelava in prikaz preizkusa kriptografskih funkcij

Preizkus zmogljivosti kriptografskih funkcij knjižnice Cryptography se zgleduje po preizkusu *openssl speed*, ki je prikazan v naslednjem podpoglavju. Test beleži posamezne vrednosti bitnega pretoka ali operacij na sekundo za vsako izbrano kriptografsko funkcijo. Izdelan je tako, da na začetku pridela pet naključno pridelanih oktetnih vrednosti določene dolžine, ki služijo kot čistopis ali vhodni podatki za različne kriptografske funkcije.

Zaradi lažje primerjave med preizkusi je dolžina čistopisov enaka kot pri preizkusu openssl speed; 16 kb, 64 kb, 256 kb, 1024 kb in 4096 kb. Za preizkus so izbrane naslednje kriptografske funkcije; SHA-256, SHA-512, RSA-1024, RSA-2048, RSA-4096, AES-128, AES-192, AES-256, ECDSA-secp256r1 in ECDSA-secp384r1.

Za ugotavljanje zmogljvosti se pri različnih funkcijah meri število operacij na sekundo, z razliko, da se pri funkcijah družine SHA-2 in AES izračuna bitna pretočnost v Mb/s, saj se tako naredi lažja primerjava z rezultati preizkusa knjižnice OpenSSL. Bitna pretočnost se izračuna kot produkt dolžine bloka v oktetih in števila operacij na sekundo.

4.1 Programska knjižnica OpenSSL

Programska knjižnica OpenSSL je zbirka kriptografskih funkcij, ki se jih uporablja v aplikacijah za zagotavljanje varnostnih lastnosti pri spletni komunikaciji. Vsebuje odprto-kodno implementacijo protokolov SSL in TLS ter nabora različnih kriptografskih funkcij v programskem jeziku C. Prav tako vsebuje programske ovojnice za nabor različnih programskih jezikov. Leta 2014 je knjižnico OpenSSL uporabljalo kar dve tretjini vseh spletnih strežnikov [5].

Za potrebe diplomske naloge se programsko knjižnico OpenSSL predstavi kot orodje, ki se ga dostopa preko vmesnika CLI[21] (*ang. Command Line Interface*). V

[21] Vmesnik ukazne vrstice

prikazani sliki 4.1 se predstavi ukaz openssl speed in primer rezultata izbrane funkcije AES-128-CBC.

```
openssl speed -evp aes-128-cbc
Doing aes-128-cbc for 3s on 16 size blocks: 96968768 aes-128-cbc's in 3.00s
Doing aes-128-cbc for 3s on 64 size blocks: 26486985 aes-128-cbc's in 3.00s
Doing aes-128-cbc for 3s on 256 size blocks: 6744643 aes-128-cbc's in 3.00s
Doing aes-128-cbc for 3s on 1024 size blocks: 1693670 aes-128-cbc's in 3.00s
Doing aes-128-cbc for 3s on 8192 size blocks: 211914 aes-128-cbc's in 2.99s
```

Slika 4.1: Izvajanje preizkusa zmogljivosti za AES-128-CBC v CLI okolju

Preizkus poteka tako, da se za različne vhodne bloke podatkov izvaja šifriranje v intervalu treh sekund. Za delovanje preizkusa je uporaba zastavice evp sicer neobvezna ampak vseeno pomembna, saj sporoči programski knjižnici OpenSSL, da deluje na procesorju, ki podpira AES-NI[22] (*ang. Advanced Encryption Standard New Instructions*). Nabor specializiranih ukazov AES-NI omogoča procesorju, da uporablja manj ukazov zbirnega jeziku za računaje funkcij AES. Kar posledično vpliva na hitrost računanja posamezne funkcije družine AES.

V sliki 4.2 vidimo končni rezultat preizkusa funkcije, ki se meri v kilobajtih. Vsi simetrični postopki knjižnice podajo rezultat v enaki obliki. Za določanje mere zmogljivosti se v primeru zgoščevalnih funkcij in simetrične kriptografije uporablja količina bitne pretočnosti.

```
The 'numbers' are in 1000s of bytes per second processed.
type             16 bytes     64 bytes    256 bytes   1024 bytes   8192 bytes
aes-128-cbc     517166.76k   565055.68k   575542.87k   578106.03k   580601.84k
```

Slika 4.2: Zmogljivosti v količini pretoka za AES-128-CBC v CLI okolju

Klic ukaza za preizkus funckije RSA-4096 je prikazan v sliki 4.3. Preizkus je sestavljen iz postopkov podpisovanja in verifikacije. Opazimo, da ne uporabimo zastavice *evp* pri podpisovanju, saj dodaten nabor ukazov AES-NI ni relevanten pri operacijah funkcij RSA.

```
openssl speed rsa4096
Doing 4096 bit private rsa's for 10s: 905 4096 bit private RSA's in 10.00s
Doing 4096 bit public rsa's for 10s: 61710 4096 bit public RSA's in 9.99s
```

Slika 4.3: Izvajanje preizkusa zmogljivosti za RSA-4096 v CLI okolju

Vsi rezultati postopkov, ki temeljijo na asimetrični kriptografiji so podani v obliki, ki je ilustrirana v sliki 4.4. Vsebujejo ime postopka, čas podpisovanja in

[22] Podaljšek x86 in x64 nabora procesorskih ukazov za računanje kriptografske funkcije AES.

verifikacije 22 oktetnega sporočila ter števila operacij na sekundo. V primeru asimetričnih kriptografskih postopkov, se za mero zmogljivosti uporablja število operacij na sekundo.

```
                  sign    verify    sign/s verify/s
rsa 4096 bits 0.011050s 0.000162s     90.5   6177.2
```

Slika 4.4: Zmogljivosti v številu operacij na sekundo za RSA-4096 v CLI okolju

4.2 Programska knjižnica Cryptography

V Odsek programske kode 7 se prikaže samo preizkus funkcije AES zaradi lažje predstavitve delovanja preizkusa. Preizkus sam po sebi tudi dovolj nazorno predstavi osnovno strukturo izdelanih preizkusov. Dodatek diplomske naloge vsebuje visoko-nivojski prikaz delovanja preizkusa v obliki diagrama. V tem delu se tudi nahaja podrobnejša razlaga parametrov preizkusa.

```
def test_AES_encrypt(self):
    data = []
    sts = StatisticTimeSeries("encrypt_timer")
    for element in self.AES_algs:
        oper, key = element[0],element[1]
        for n, input_blck in enumerate(self.data_collection):
            sts.start('%s: block size %s bytes' % (oper, str(self.test_parameters["input"][n])),
                      bytes=self.test_parameters["input"][n])
            for i1 in range(self.test_parameters["iteration"]):
                sts.new()
                for i2 in range(self.test_parameters["num_of_op"]):
                    self.AES_encrypt(key, input_blck)
                sts.add(self.test_parameters["num_of_op"])
            data.append(sts.stop())

    transp = list(zip(*data))
    self.save_results("AES", self.AES_algs, transp)
```

Odsek programske kode 7: Preizkus zmogljivosti AES za knjižnico Cryptography

Vsi preizkusi uporabljajo po meri izdelan modul *StatisticTimeSeries*, ki meri čas potreben za izvedbo določene funkcije. Modul poskrbi, da so meritve točne, saj je končna vrednost izbrana kot povprečje stotih meritev.

Za vsak izbran AES algoritem se iz seznama element pridobi ime operacije (*oper*) in ključ (*key*), ki je v nadaljevanju uporabljen pri različnih čistopisih (vhodnih vrednostih) že prej omenjene velikosti. Pri vsakem vhodnem bloku uporabimo nov časovni merilnik, ki mu dodelimo posebno ime sestavljeno iz vrednosti *oper* in dolžine posameznega vhodnega bloka. V naslednjem koraku določimo, da se test ponovi tolikokrat kakor je določeno na začetku preizkusa v *num_of_iter* spremenljivki. Na koncu se rezultati preizkusa shranijo v seznamu *data*, ki pa se za tem transponira in shrani v JSON[23] datoteko na trdem disku preko funkcije *save_results*. Funkcija kot vrednost rezultata shrani seznam rezultat števila operacij na sekundo za vsak posamezni vhodni blok podatkov in vsako izmed določenih funkcij AES.

Edina razlika, ki se pojavi med različnimi preizkusi, je v imenu klicane kriptografske funkcije v jedru preizkusa.

4.3 Prikaz zmogljivosti in učinkovitosti kriptografskih funkcij

Kot je že omenjeno v prejšnjih poglavjih, se za potrebe naloge, v primeru preizkusa funkcij SHA-2 in AES, zmogljivost definirana kot količina bitnega pretoka, ki je podana v *MB/s*. V primeru preizkusa funkcij RSA in ECDSA pa se zmogljivost definira kot število operacij na sekundo.

Mera bitne pretočnosti se izračuna preko enačbe 4.1, kjer predstavljajo simboli; P_{blok} - količino bitnega pretoka, O - število izvedenih operacij posamezne funkcije, L_{blok} - dolžino posameznega bloka vhodnih podatkov.

$$P_{blok}\left[\frac{B}{s}\right] = O\left[\frac{op}{s}\right] * L_{blok}\,[B] \tag{4.1}$$

V nadaljevanju so prikazani rezultati preizkusa zmogljivosti knjižnic Cryptography in OpenSSL na strojni in programski opremi, ki je določena v uvodnem poglavju. Pri rezultatih merjenja bitne pretočnosti, so rezultati prikazani na grafih. Na abscisni osi je prikazana velikost vhodnih blokov v oktetih, na ordinatni osi pa bitna pretočnost v megabajtih na sekundo. Pri rezultatih merjenja operacij na sekundo, pa je zmogljivost funkcij prikazana v tabelah.

V naslednjih podpoglavjih so prav tako prikazane mere učinkovitosti predstavljenih kriptografskih funkcij. Učinkovitost je definirana kot razmerje vrednosti bitne pretočnosti ali operacij na sekundo med funkcijami iz knjižnice Cryptography in OpenSSL.

[23] Datotečni tip odprtega standarda, ki vsebuje človeško-berljiv tekstovni format za prenašanje podatkovnih objektov.

Izračuna se z enačbo 4.2, kjer predstavljajo simboli; η - učinkovitost, ${P^{pyc}}_{blok}$ - bitno pretočnost posameznega bloka in funkcije iz knjižnice Cryptography, ${P^{ossl}}_{blok}$ - bitno pretočnost posameznega bloka in funkcije iz knjižnice OpenSSL.

$$\eta[\%] = {P^{pyc}}_{blok}\left[\frac{MB}{s}\right] / {P^{ossl}}_{blok}\left[\frac{MB}{s}\right] \tag{4.2}$$

Vse kriptografske funkcije imajo začetne režijske stroške, ki pojasnijo počasnejše bitne pretočnosti pri manjših vhodnih blokih. Režijske stroške večinoma sestavlja računanje ključnega razporeda[24] (ang. key-scheduling), inicializacija funkcij in dodelava spomina (*ang. memory allocation*).

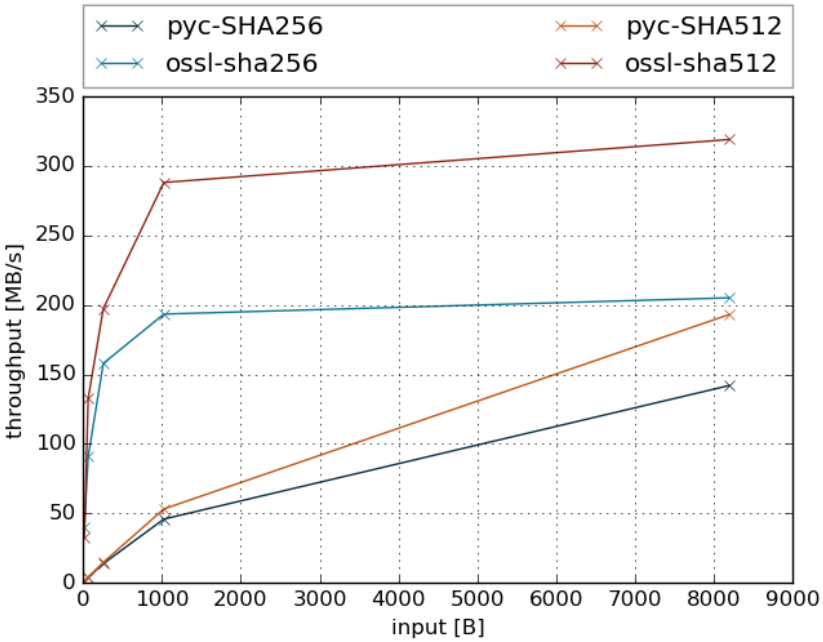
Imena predstavljenih krivulj so sestavljena iz predpone, ki predstavlja knjižnico; *pyc* (Python Cryptography) ali *ossl* (OpenSSL) in kriptografske funkcije.

4.3.1 Funkcije SHA

Rezultati izračunov bitne pretočnosti in učinkovitosti programske knjižnice so predstavljeni v dveh slikah. V sliki 4.5 opazimo razliko med zmogljivostjo knjižnice OpenSSL in Cryptography. Razlika je smiselna, saj funkcije knjižnice Cryptography v zaledju uporabljajo OpenSSL.

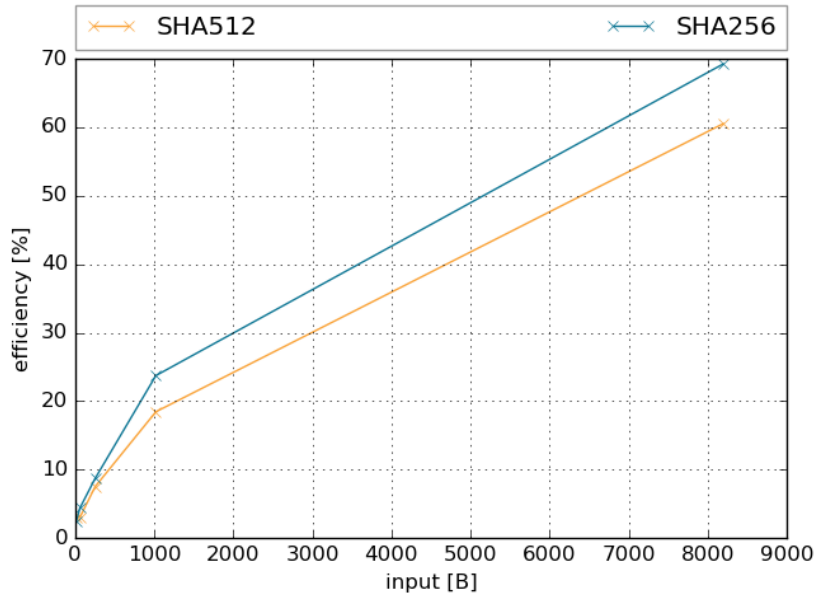
Zanimiva je neposredna primerjava med SHA-256 in SHA-512 funkcijami, saj je SHA-512 občutno hitrejša v primerjavi s SHA-256, čeprav SHA-512 opravi 25% več iteracij pri računanju zgoščene vrednosti. Razlika se pojavi zaradi uporabljene strojne in programske opreme (64-bitni procesor in operacijski sistem), saj SHA-512 uporablja pri računanju 64 bitne besede, SHA-256 pa 32 bitne besede [6]. V idealnih pogojih je SHA-512 lahko do 1,6-krat hitrejša od SHA-256, tako kot je pri knjižnici OpenSSL. Zaradi režijskih stroškov pa je razlika lahko tudi občutno manjša, kot je pri knjižnici Cryptography.

[24] Proces razširjanja krajšega ključa v več različnih ključev, ki so uporabljeni v posameznih iteracijah.



Slika 4.5: Zmogljivosti v količini pretoka za SHA-2

V sliki 4.6 je predstavljena učinkovitost zgoščevalnih funkcij družine SHA-2 za programsko knjižnico Cryptography. Opazimo, da je učinkovitost funkcije SHA-256 višja od funkcije SHA-512. Čeprav v teoriji SHA-512 opravi več ponovitev pri računanju zgoščene vrednosti kot SHA-256, je zaradi izbrane strojne (64-bitni procesor) in programske (64-bitni operacijski sistem) opreme prikazana kot manj učinkovitejša funkcija pri vseh vhodnih blokih.

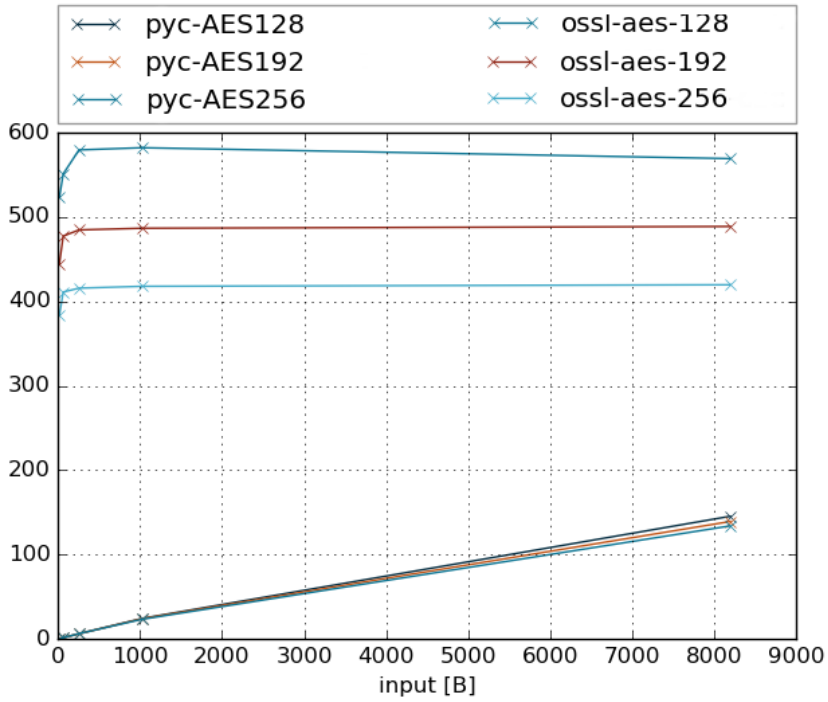


Slika 4.6: Izračun učinkovitosti za SHA-2

4.3.2 Funkcije AES

V sliki 4.7 opazimo očitno razliko med bitno pretočnostjo funkcij obeh knjižnic. Zmogljivosti različnih funkcij AES iz knjižnice Cryptography, se med seboj skoraj nič ne razlikujejo, kar nakazuje na dejstvo, da je zmogljivost posamezne pyc-AES funkcije najbolj odvisna od režijskih stroškov, ki nastanejo pri klicu zaledne knjižnice. Tega pojava pri funkcijah knjižnice OpenSSL ne opazimo, saj je bitna pretočnost funkcij v večji meri odvisna od računske zahtevnosti in ne režijskih stroškov računanja.

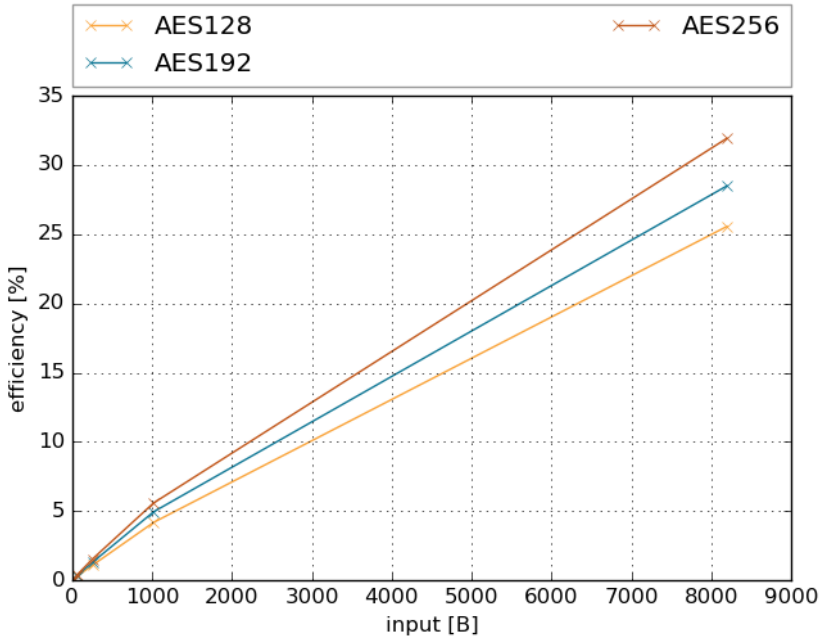
Programska knjižnica OpenSSL je optimizirana na procesorjih Intel in pri računanju funkcij AES podpira AES-NI.



Slika 4.7: Zmogljivosti v količini pretoka za AES

V sliki 4.8 vidimo kako se razlikujejo šifrirni postopki AES med seboj po učinkovitosti. Bolj računsko zahtevni postopki imajo višjo učinkovitost, saj se večino procesorskega časa porabi za računanje in ne za obdelavo režijskih stroškov pri klicu nižje ležečih kriptografskih funkcij.

Opazimo tudi, da se učinkovitost povečuje z velikostjo bloka vhodnih podatkov, kar se sklada s predpostavko, da je učinkovitost odvisna od obdelave režijskih stroškov.



Slika 4.8: Izračun učinkovitosti za AES

4.3.3 Funkcije RSA

Primerjava rezultatov iz tabele 4.1 nam potrdi trditev, da se pri funkcijah, ki so računsko bolj intenzivne, pojavi manj razlik v številu opravljenih operacij na sekundo med funkcijami OpenSSL in Cryptography knjižnic.

Čeprav v teoriji ni razlik med RSA verifikacijskim in postopkom podpisovanja, se zaradi znanih ranljivosti [13] uporabljajo določeni parametri (manjši javni eksponent, večji zasebni eksponent), ki vplivajo na hitrost RSA kriptografskih postopkov. Kar pa pomeni, da je verifikacijski RSA postopek veliko hitrejši od podpisovanja.

	Verifikacija	Podpisovanje
OpenSSL-RSA-1024	76638	5239
pyc-RSA-1024	16626	3141
OpenSSL-RSA-2048	18945	633
Pyc-RSA-2048	10836	598
OpenSSL-RSA-4096	6218	90
Pyc-RSA-4096	4586	89

Tabela 4.1: Zmogljivosti v številu operacij na sekundo za RSA

Rezultati predstavljeni v tabeli 4.2 močno poudarijo že omenjeno dejstvo, da so bolj učinkovite kriptografske funkcije, tudi računsko intenzivnejše. To je še posebej razvidno pri RSA postopku s 4096 bitnim ključem, saj je pri podpisovanju učinkovitost skoraj sto odstotna.

	Verifikacija	Podpisovanje
RSA-1024	21%	59%
RSA-2048	57%	94%
RSA-4096	73%	98%

Tabela 4.2: Izračun učinkovitosti za RSA

4.3.4 Funkcije ECDSA

Tabela 4.3 poda vrednosti verifikacije in podpisovanja ECDSA algoritma dveh različnih eliptičnih krivulj. Opazimo, da je za razliko od RSA algoritma, podpisovanje hitrejše od verifikacije. To je zaradi uporabe manj bitnih števila, pri računanju končne vrednosti, kar je glavna prednost algoritma ECDSA pred algoritmom RSA.

	Verifikacija	Podpisovanje
OpenSSL-ECDSA-secp256r1	2904	7005
Pyc-ECDSA-secp256r1	2594	4825
OpenSSL-ECDSA-secp384r1	996	4125
Pyc-ECDSA-secp384r1	927	1052

Tabela 4.3: Zmogljivosti v številu operacij na sekundo za ECDSA

V tabeli 4.4 imamo naslednjo zbirko rezultatov za učinkovitost funkcije ECDSA. Vidimo, da je učinkovitost postopka višja pri verifikacijskem postopku, za razliko od postopkov RSA, ki pa imajo višjo učinkovitost pri podpisovanju.

	Verifikacija	Podpisovanje
ECDSA-secp256r1	89%	68%
ECDSA-secp384r1	93%	25%

Tabela 4.4: Izračun učinkovitosti za ECDSA

4.4 Ugotovitve

Rezultati preizkusa zmogljivosti so pokazali surove vrednosti različnih kriptografskih funkcijah obeh knjižnic. Primerjava teh vrednosti nazorno prikaže hitrost funkcij implementiranih neposredno v programskem jeziku C (OpenSSL), za razliko od posredne implementacije, ki deluje kot API v programskem jeziku Python (Cryptography).

Izračun učinkovitosti nam omogoča uporabo dodatne metrike pri vrednotenju knjižnice Cryptography. Nakazuje na dejstvo, da je uporaba knjižnice Cryptography bolj smiselna v primerih, kjer se operira z večjimi bloki vhodnih podatkov, in kjer se uporablja računsko zahtevnejše kriptografske postopke.

Izbira knjižnice je odvisna od aplikacije in programskega jezika. Vendar je lažje opravičiti stroške režije, ki nastanejo pri uporabi knjižnice kot je Cryptography, če uporabljamo računsko zahtevnejše kriptografske funkcije in večje vhodne bloke podatkov. Prav tako, ilustrirani rezultati bitne pretočnosti in operacij na sekundo prikazujejo, da se pri določenih funkcijah kot je RSA-4096, hitrost izvajanja postopka preko programske ovojnice knjižnice Cryptography približa zaledni implementaciji v knjižnici OpenSSL.

Zagotavljanje varnostnih lastnosti na nivoju HTTP[25] (*ang. Hyper Text Transfer Protocol*) zahtev in odgovorov opravlja protokol TLS, ki ga lahko implementirano tudi s pomočjo programske knjižnice Cryptography. Povprečna dolžine paketa ali glave (*ang. header*) zahtev in odgovorov HTTP je okoli 800 oktetov [14], iz preizkusa pa je razvidno, da knjižnica Cryptography ni najučinkovitejša ravno v razponu do 1000 oktetov.

Pomembno je, da se zavedamo omejitev programskih knjižnic, ki delujejo kot programske ovojnice, saj je hitrost izvajanja funkcij v neposrednih implementacijah zmeraj višja. Proces njihove posvojitve in uporabe je ponavadi odvisen od drugih faktorjev kot platforma uporabe, tip aplikacije, izbira programskega jezika, itd.

[25] Protokol za prenašanje informacij po svetovnem spletu.

5 Zaključek

Predstavitev izbranih kriptografskih funkcij knjižnice Cryptography nam prikaže estetiko in preprostost uporabe programske ovojnice, ki je tako značilna za programski jezik Python. Zaradi hitre rasti in razširjenosti uporabe jezika Python pri razvijalcih spletnih storitev in spletne infrastrukture je potrebno raziskati njegov potencial tudi pri kriptografskih aplikacijah.

Cryptography v zaledju temelji na priljubljeni programski knjižnici OpenSSL in je zaradi tega pri strogi primerjavi zmogljivosti in hitrosti počasnejša, vendar pa je primerjava še vedno koristna. Primerjava nakazuje na dejstvo, da se v določenih aplikacijah programska knjižnica Cryptography približa hitrosti neposredne implementacije kriptografskih funkcij v OpenSSL. To se predvsem zgodi pri funkcijah, ki so bolj računsko zahtevne.

Glavna prednost knjižnice Cryptography izhaja iz relativne preprostosti uporabe in dejstva, da deluje kot ovojnica za različne, že priznane kriptografske knjižnice. To pomeni, da se lahko razvijalec seznani samo s Cryptography pri razvijanju kriptografskih rešitev na različnih arhitekturah.

V praksi, se Cryptography pričenja uveljavljati kot standardna kriptografska knjižnica za programski jezik Python, zato je izdelan preizkus dragoceno orodje pri določanju primernosti implementacije določenih kriptografskih funkcij. Poleg tega, nam ugotovitve omogočajo tudi objektivno sklepanje o uporabi knjižnice Cryptography pri različnih aplikacijah varovanja komunikacij ali shranjenih podatkov.

Literatura

[1] OpenSSL Foundation, "OpenSSL", Openssl.org, 2016. [Na spletu]. Dosegljivo: https://www.openssl.org/docs/.

[2] "Welcome to Cryptography — Cryptography 1.5.dev1 documentation", Cryptography.io, 2016. [Na spletu]. Dosegljivo: https://Cryptography.io/en/latest/.

[3] "Speedtest and Comparsion of Open-Source Cryptography Libraries and Compiler Flags - panthema.net", Panthema.net, 2016. [Na spletu]. Dosegljivo: https://panthema.net/2008/0714-Cryptography-speedtest-comparison/.

[4] Cherdantseva Y. and Hilton J.: "Information Security and Information Assurance. The Discussion about the Meaning, Scope and Goals". In: Organizational, Legal, and Technological Dimensions of Information System Administrator. Almeida F., Portela, I. (eds.). IGI Global Publishing. (2013)

[5] "Critical crypto bug in OpenSSL opens two-thirds of the Web to eavesdropping", Ars Technica, 2016. [Na spletu]. Dosegljivo: http://arstechnica.com/security/2014/04/critical-crypto-bug-in-openssl-opens-two-thirds-of-the-web-to-eavesdropping/.

[6] “SECURE HASH STANDARD”, Federal Information Processing Standards Publication 180-3, United States National Institute of Standards and Technology (NIST). October 2008, http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf

[7] "Announcing the ADVANCED ENCRYPTION STANDARD (AES)" (PDF). Federal Information Processing Standards Publication 197. United States National Institute of Standards and Technology (NIST). November 26, 2001. Retrieved October 2, 2012.

[8] Rivest, R.; Shamir, A.; Adleman, L. (February 1978). "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". Communications of the ACM 21 (2): 120–126. doi:10.1145/359340.359342.

[9] "Mask Generation Functions | Intel® Software", *Software.intel.com*, 2016. [Na spletu]. Dosegljivo: https://software.intel.com/en-us/node/503159. [Accessed: 15- Aug- 2016].

[10] "ECDSA: The digital signature algorithm of a better internet", CloudFlare, 2014. [Na spletu]. Dosegljivo: https://blog.cloudflare.com/ecdsa-the-digital-signature-algorithm-of-a-better-internet/.

[11] “DIGITAL SIGNATURE STANDARD”, Federal Information Processing Standards Publication 186-2, United States National Institute of Standards and Technology (NIST). Januar 2000, http://csrc.nist.gov/publications/fips/fips180-3/fips180- 3_final.pdf

[12] Daniel J. Bernstein, Tanja Lange. "Security dangers of the NIST curves." May 2013.

[13] M. Wiener. Cryptanalysis of short RSA secret exponents. IEEE Transactions on Information Theory, 36:553-558, 1990

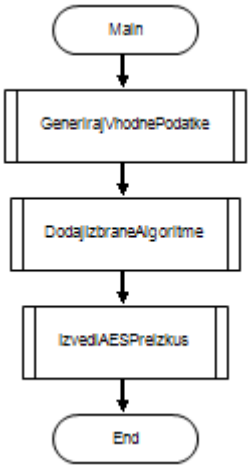
[14] "SPDY: An experimental protocol for a faster web - The Chromium Projects", Dev.chromium.org, 2016. [Na spletu]. Dosegljivo: http://dev.chromium.org/spdy/spdy-whitepaper.

A Diagram poteka za preizkus zmogljivosti funkcij AES

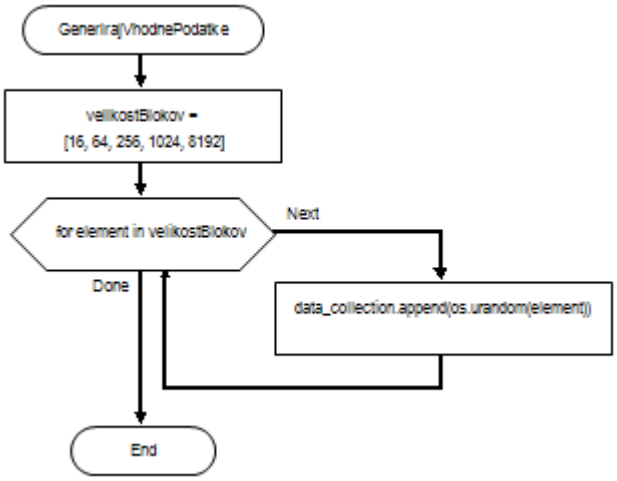
V naslednjih slikah je predstavljen visoko-nivojski prikaz delovanja različnih delov preizkusa zmogljivosti za kriptografsko funkcijo AES, ki je prav tako predstavljena v glavnem delu naloge.

Za delovanje preizkusa se v čistem začetku inicializirajo vrednosti spremenljivk, ki so ključne za delovanje preizkusa; *data_collection*, *operation*, *AES_algs*. Vse naštete spremenljivke so seznami, katerih vloga je opisana in prikazana spodaj. Zadnji dve; *num_of_iter, num_of_op* pa sta celoštevilni vrednosti, ki sta ključni za delovanje knjižnice StatisticTimeSeries - sts. Časovnik sts izmeri povprečno vrednost časovnega izvajanja ene funkcije na podlagi večih meritev katerih določata zadnja parametra.

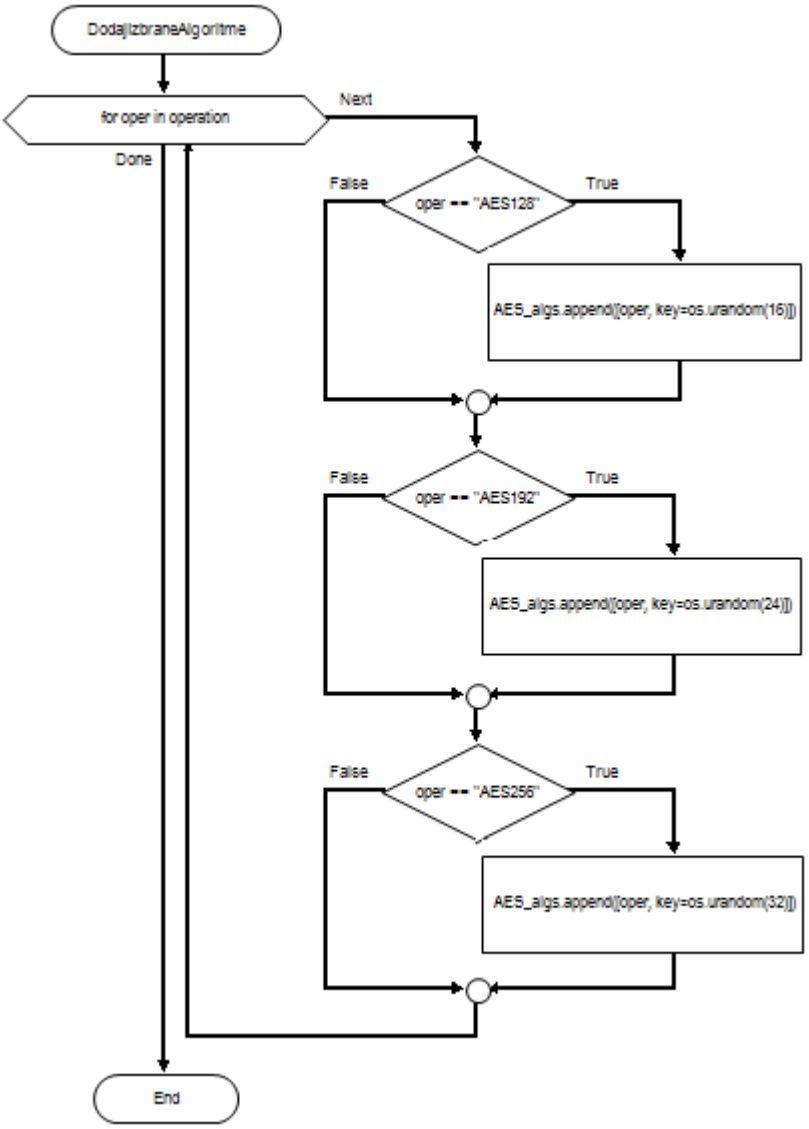
- *data_collection*: shrani naključno pridelane vhodne vrednosti, s pomočjo funkcije *os.urandom()* funkcije. Vrednosti služijo kot čistopis za kriptografske postopke.
- *operation*: hrani imena postopkov, ki se jih preizkuša, npr. ["AES-192", "AES-256"]
- *AES_algs*: hrani elemente, ki so seznami, sestavljeni iz imena izbranih postopkov in vrednosti naključno pridelanega zasebnega ključa primerne dolžine, npr. [["AES-192", *os.urandom(24)*], ["AES-256", *os.urandom(32)*]]
- *num_of_iter*: hrani število iteracij ali ponovitev, ki bodo izvedene za vsak izbran preizkus. V našem primeru je nastavljeno na 100 iteracij.
- *num_of_op*: Parameter, ki igra podobno vlogo kot *num_of_iter*, in omogoča, da določimo večkratno izvajanje vsake posamezne iteracije. Funkcionalnost časovnika nam omogoča dodatne statistične analize s pridobljenimi časovnimi vrednosti, ki pa za potrebe naloge niso bile uporabljene, zato je vrednost nastavljena na 1.



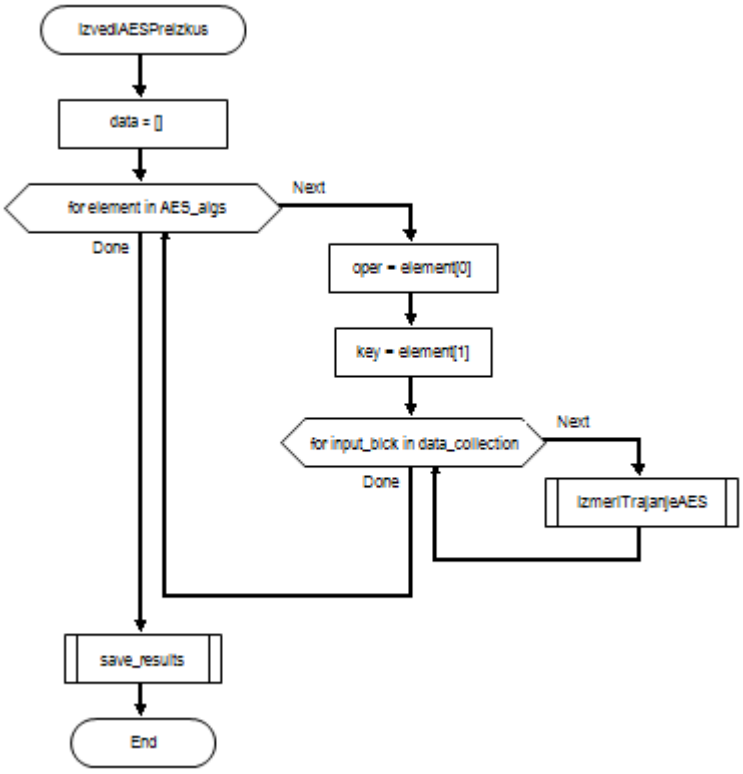
Slika A.1: Vrstni red izvajanja preizkusa za funkcije AES



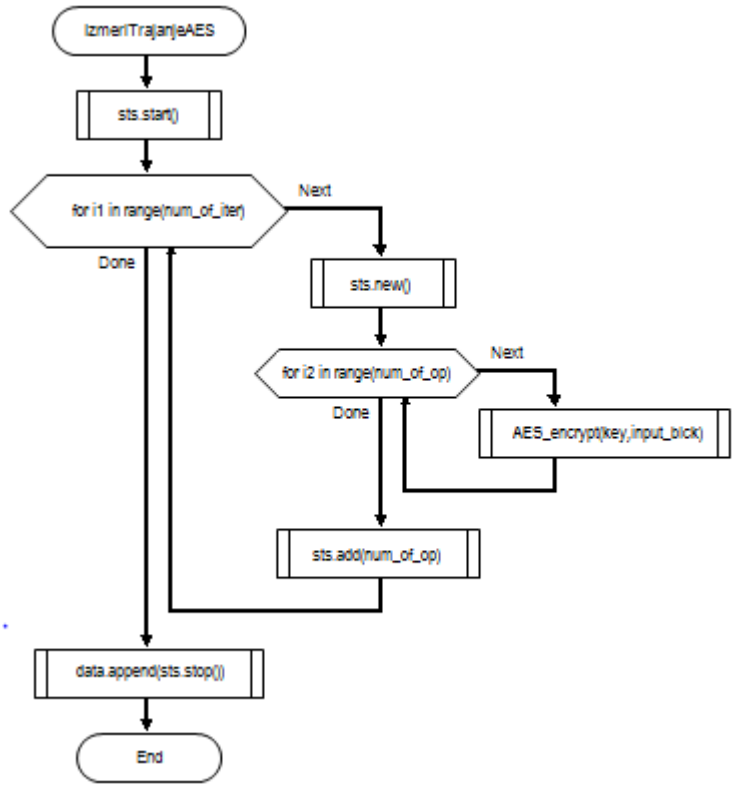
Slika A.2: Diagram GenerirajVhodnePodatke



Slika A.3: Diagram DodajIzbraneAlg



Slika A.4: Diagram IzvediPreizkusAES



Slika A.5: Diagram IzmeriTrajanjeAES